Polymorphism

Objectives

- Using an “instanceof” operator
- Polymorphism
- Interface
- Abstract class

Using an "instanceof" operator

- Dynamic and Static type
  - dynamic type: A reference variable that has the type of the superclass can store the address of the object of sub class. It is called to be *dynamic type*, the type that is has at runtime.

    Rectangle obj1 = new Box();
  - Static type: The type that it has when first declared. Static type checking is enforced by the compiler.

    Box obj2 = new Box();
- *"Instanceof" operator:* It checks whether the reference of an object belongs to the provided type or not, the instanceof operator will return true or false.

```
If ( obj1  instanceof  Box)
      System.out.println(" obj1 is pointing to the Box object");
```

Casting

- A variable that has the type of the superclass only calls methods of the superclass. To call methods of the subclass we must *cast explicitly*
- *for example,*

```
Rectangle obj = new Box();
((Box)obj).setHeight(300);
```

Polymorphism

Polymorphism relates the implementation for an object based on its type

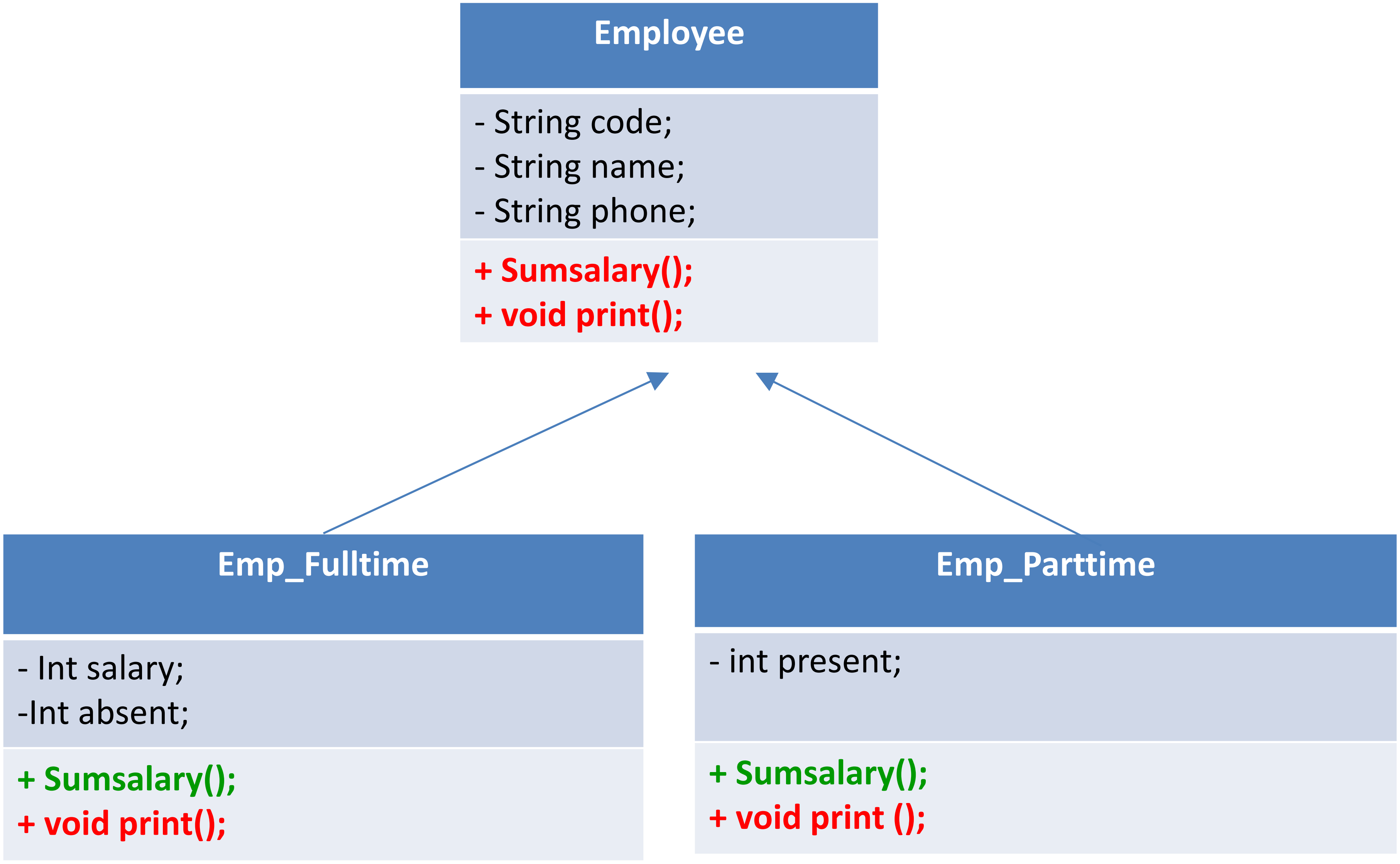
Polymorphism

Ability allows many versions of a method based on overloading and overriding methods techniques.

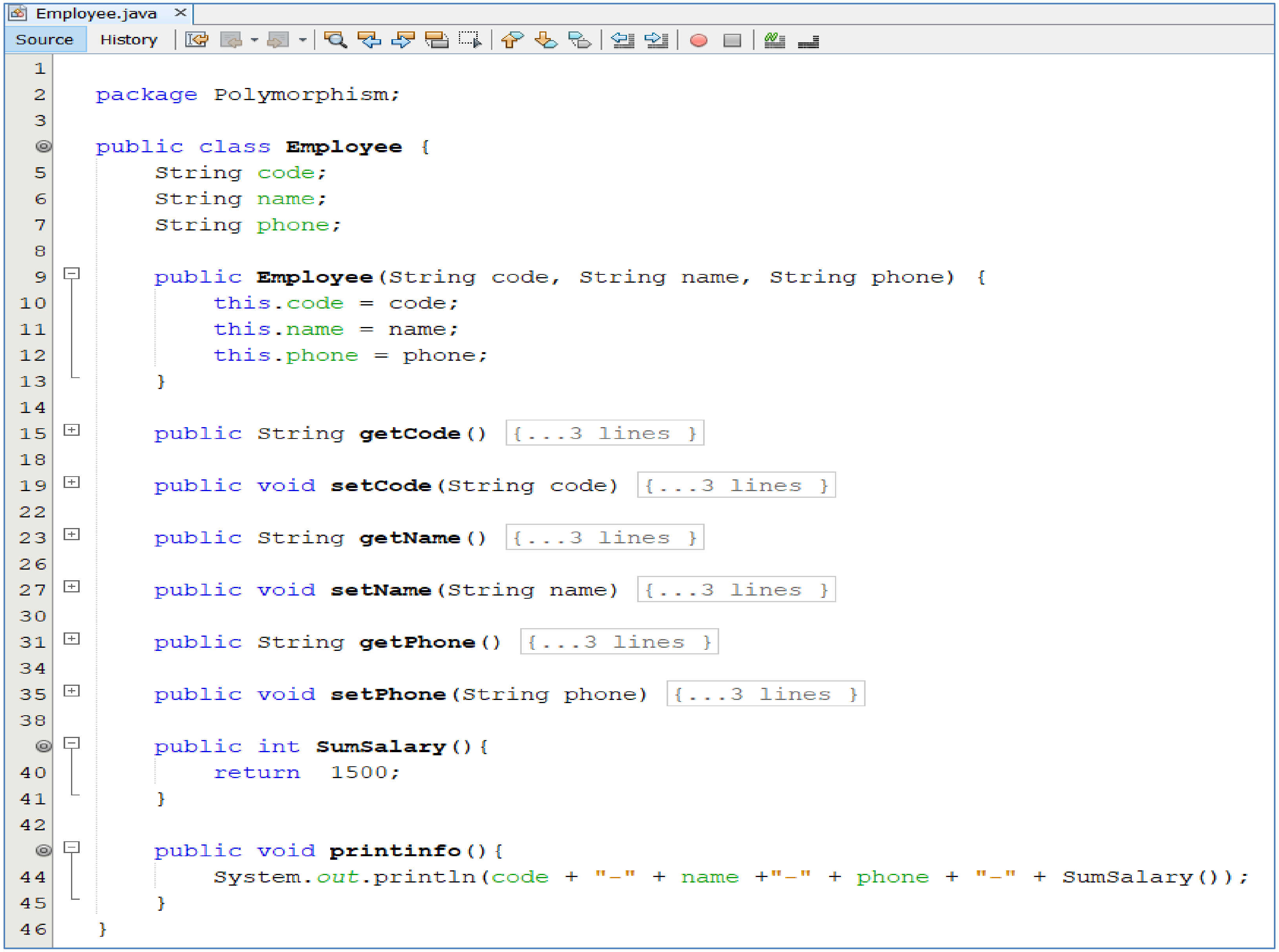
Overloading: A class can have some methods which have the same name but their parameter types are different.

Overriding: A method in the father class can be overridden in its derived classes (body of a method can be replaced in derived classes).

Polymorphism…)



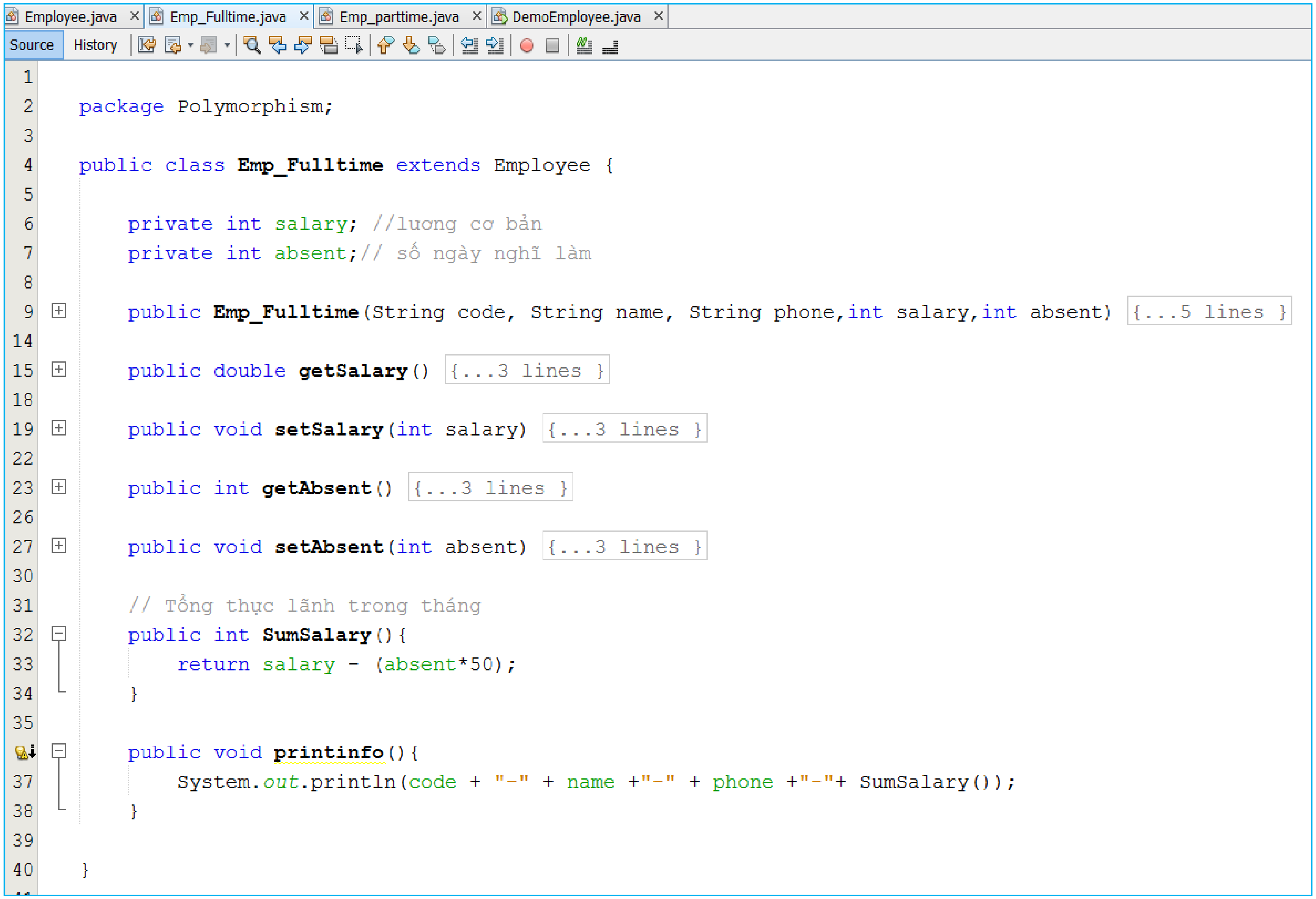
Polymorphism



Employee.java
Source History

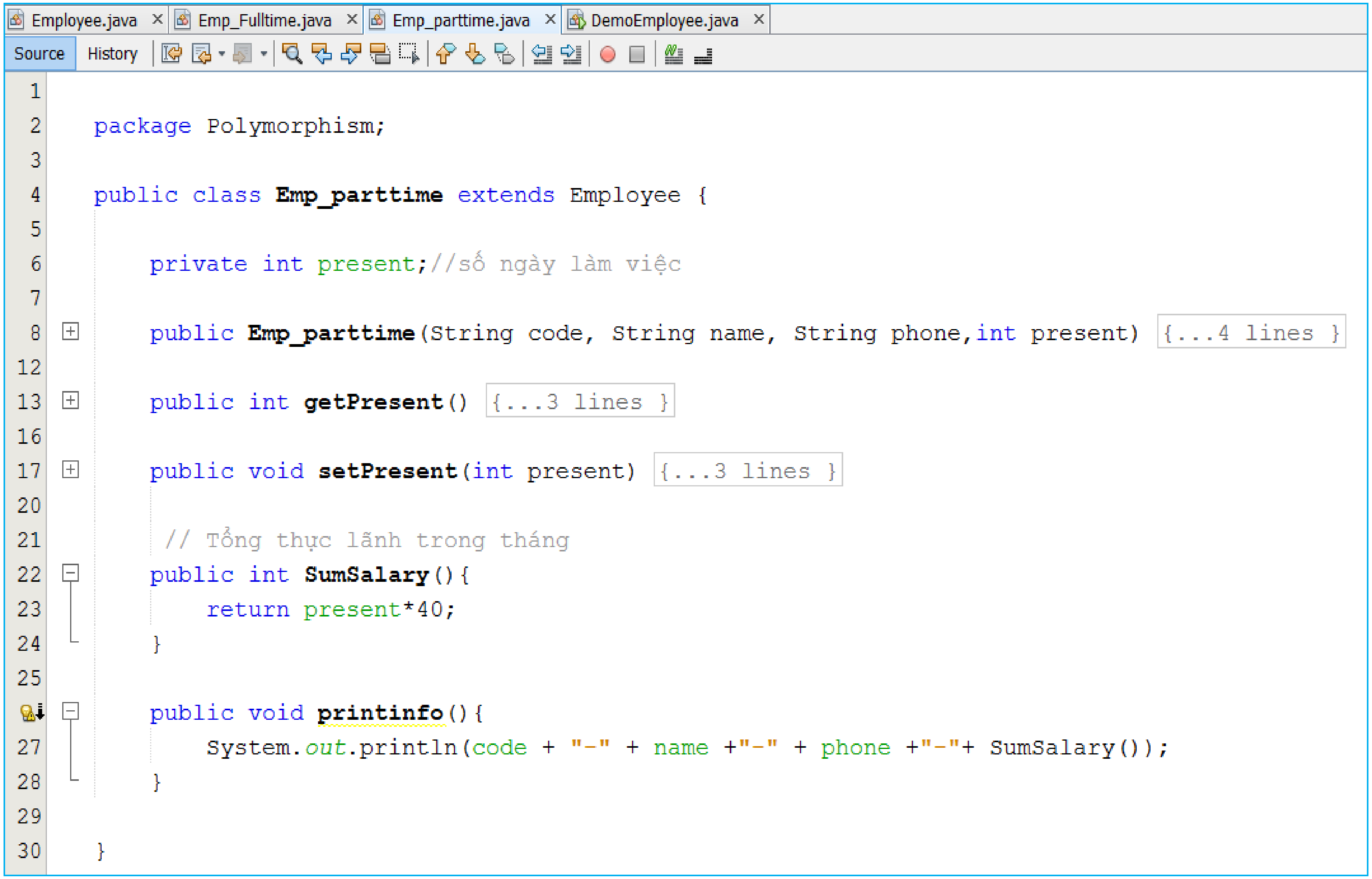
```java

package Polymorphism;

public class Employee {
    String code;
    String name;
    String phone;

    public Employee(String code, String name, String phone) {
        this.code = code;
        this.name = name;
        this.phone = phone;
    }

    public String getCode() {...3 lines }

    public void setCode(String code) {...3 lines }

    public String getName() {...3 lines }

    public void setName(String name) {...3 lines }

    public String getPhone() {...3 lines }

    public void setPhone(String phone) {...3 lines }

    public int SumSalary(){
        return  1500;
    }

    public void printinfo(){
        System.out.println(code + "-" + name +"-" + phone + "-" + SumSalary());
    }
}
```

Polymorphism



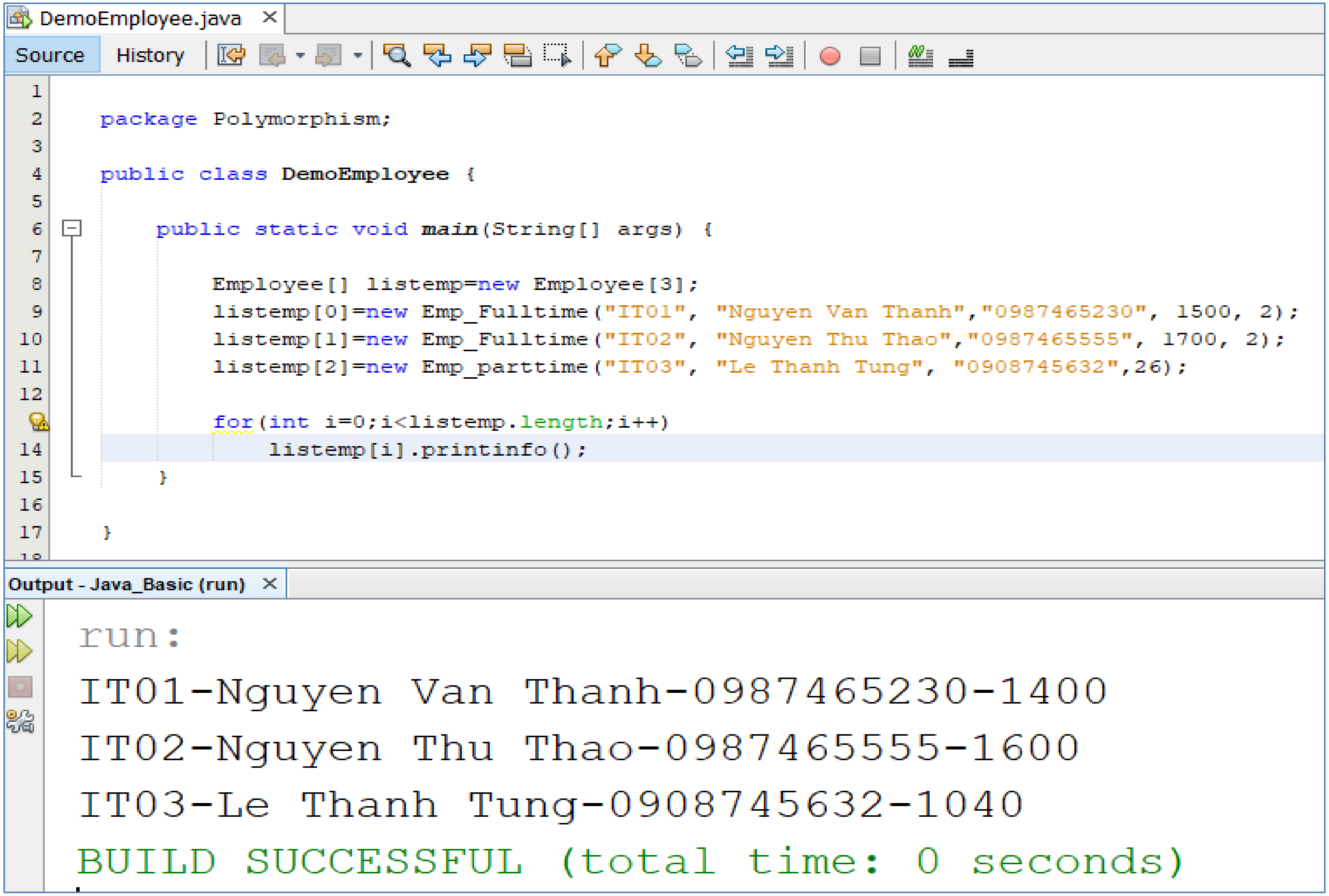
```java

package Polymorphism;

public class Emp_Fulltime extends Employee {

    private int salary; //lương cơ bản
    private int absent;// số ngày nghĩ làm

    public Emp_Fulltime(String code, String name, String phone,int salary,int absent) {...5 lines }

    public double getSalary() {...3 lines }

    public void setSalary(int salary) {...3 lines }

    public int getAbsent() {...3 lines }

    public void setAbsent(int absent) {...3 lines }

    // Tổng thục lãnh trong tháng
    public int SumSalary(){
        return salary - (absent*50);
    }

    public void printinfo(){
        System.out.println(code + "-" + name +"-" + phone +"-"+ SumSalary());
    }

}
```

Polymorphism



```
package Polymorphism;

public class Emp_parttime extends Employee {

    private int present;//số ngày làm việc

    public Emp_parttime(String code, String name, String phone,int present) {...4 lines }

    public int getPresent() {...3 lines }

    public void setPresent(int present) {...3 lines }

     // Tổng thực lãnh trong tháng
    public int SumSalary(){
        return present*40;
    }

    public void printinfo(){
        System.out.println(code + "-" + name +"-" + phone +"-"+ SumSalary());
    }

}
```

Polymorphism



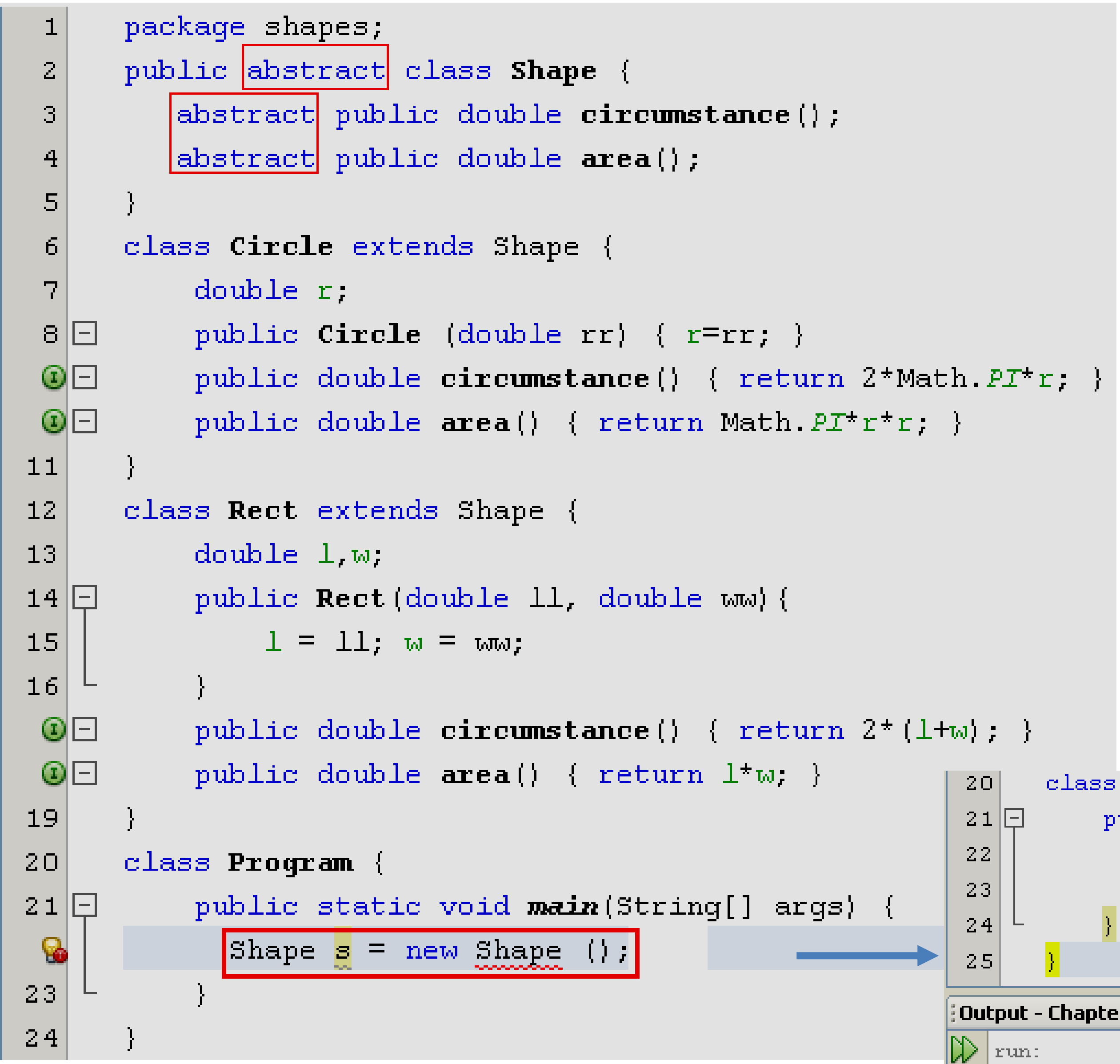
```java
package Polymorphism;

public class DemoEmployee {

    public static void main(String[] args) {

        Employee[] listemp=new Employee[3];
        listemp[0]=new Emp_Fulltime("IT01", "Nguyen Van Thanh","0987465230", 1500, 2);
        listemp[1]=new Emp_Fulltime("IT02", "Nguyen Thu Thao","0987465555", 1700, 2);
        listemp[2]=new Emp_parttime("IT03", "Le Thanh Tung", "0908745632",26);

        for(int i=0;i<listemp.length;i++)
            listemp[i].printinfo();
    }

}
```

```
run:
IT01-Nguyen Van Thanh-0987465230-1400
IT02-Nguyen Thu Thao-0987465555-1600
IT03-Le Thanh Tung-0908745632-1040
BUILD SUCCESSFUL (total time: 0 seconds)
```

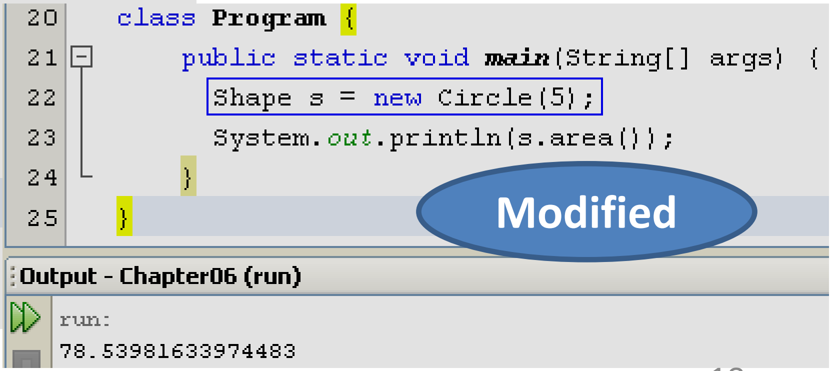
Abstract Classes

- Used to define ***what* behaviors a class is required to perform without having to provide an explicit implementation.**
- **It is the result of so-high generalization**
- **Syntax to define a abstract class**
  - *public abstract class className{ ... }*
- It isn't necessary for all of the methods in an abstract class to be abstract.
- An abstract class can also declare implemented methods.

Abstract Classes…



```java
package shapes;
public abstract class Shape {
    abstract public double circumstance();
    abstract public double area();
}
class Circle extends Shape {
      double r;
      public Circle (double rr) { r=rr; }
      public double circumstance() { return 2*Math.PI*r; }
      public double area() { return Math.PI*r*r; }
}
class Rect extends Shape {
      double l,w;
      public Rect(double ll, double ww){
          l = ll; w = ww;
      }
      public double circumstance() { return 2*(l+w); }
      public double area() { return l*w; }
}
class Program {
      public static void main(String[] args) {
          Shape s = new Shape ();
      }
}
```



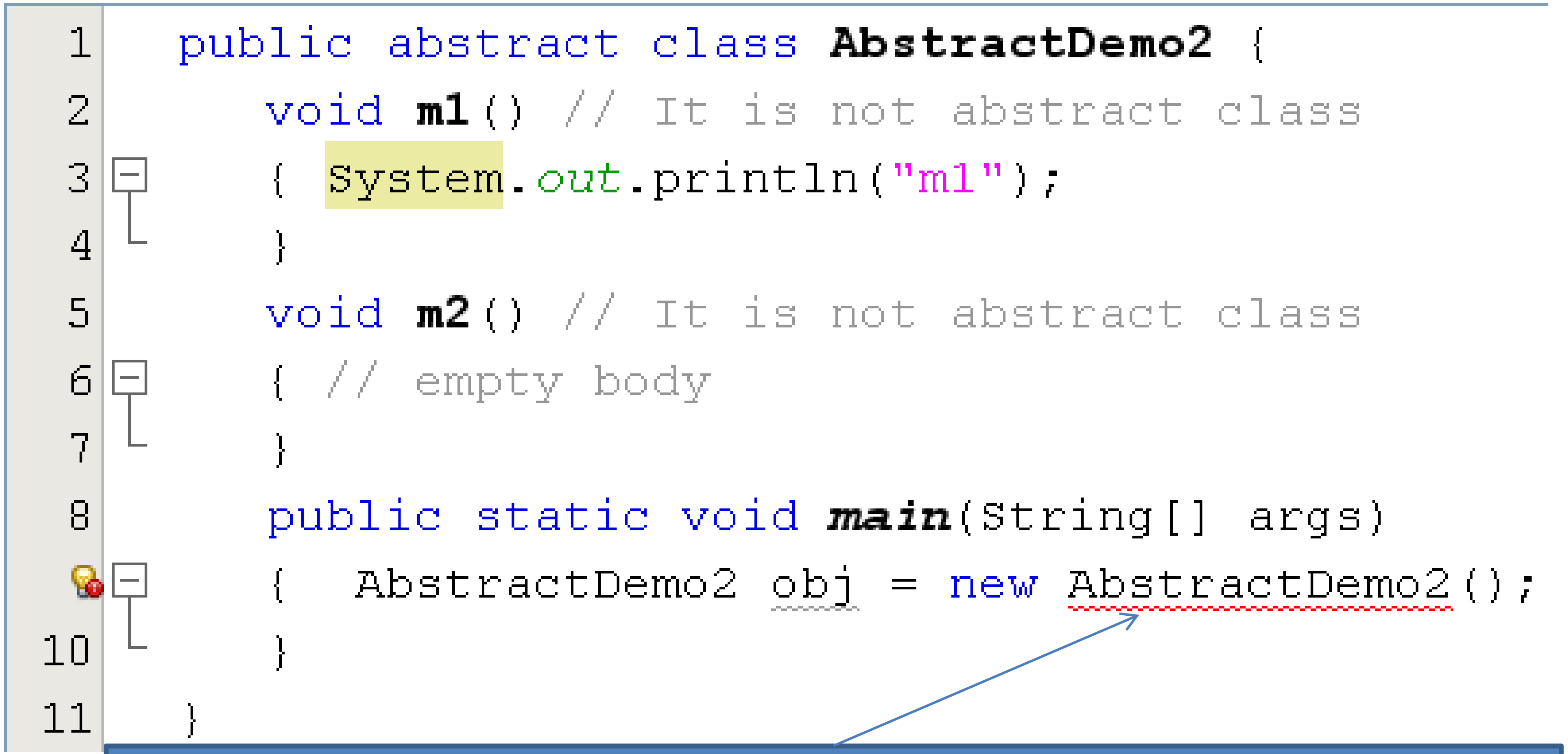
```java
class Program {
    public static void main(String[] args) {
        Shape s = new Circle(5);
        System.out.println(s.area());
    }
}
```

Modified

Output - Chapter06 (run)

```
run:
78.53981633974483
```

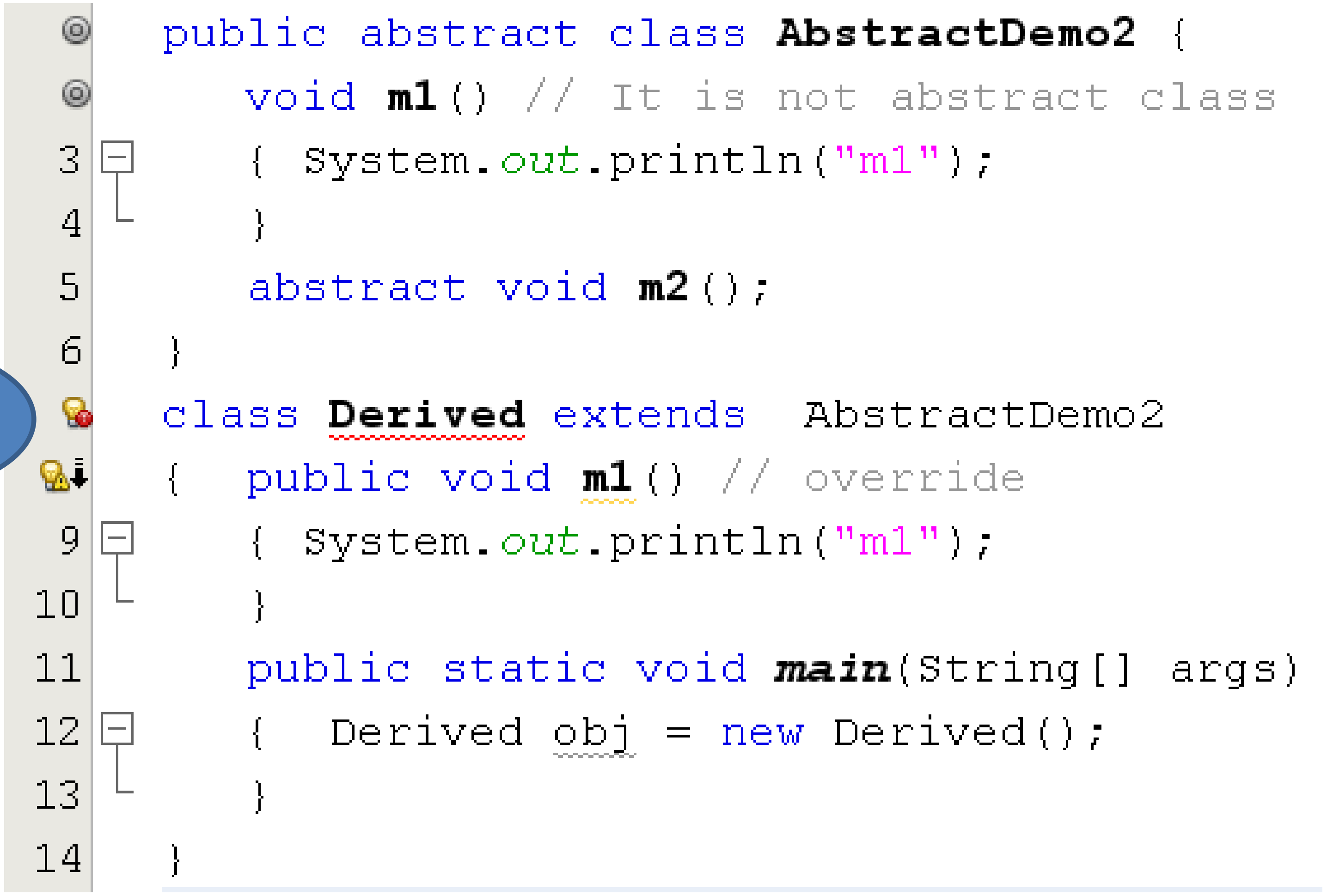
Abstract Classes…



```
public abstract class AbstractDemo2 {
    void m1() // It is not abstract class
    { System.out.println("m1");
    }
    void m2() // It is not abstract class
    { // empty body
    }
    public static void main(String[] args)
    {   AbstractDemo2 obj = new AbstractDemo2();
    }
}
```

This class have no abstract method but it is declared as an abstract class. So, we can not initiate an object of this class.

Abstract Classes…



```
public abstract class AbstractDemo2 {
    void m1() // It is not abstract class
    { System.out.println("m1");
    }
    abstract void m2();
}
class Derived extends  AbstractDemo2
{   public void m1() // override
    { System.out.println("m1");
    }
    public static void main(String[] args)
    {   Derived obj = new Derived();
    }
}
```

Error.
Why?

Implementing Abstract Methods

- Derive a class from an abstract superclass, the subclass will inherit all of the superclass's features, all of ***abstract* methods** included**.**
- To replace an inherited abstract method with a concrete version, the subclass need merely override it.
- Abstract classes ***cannot be instantiated***

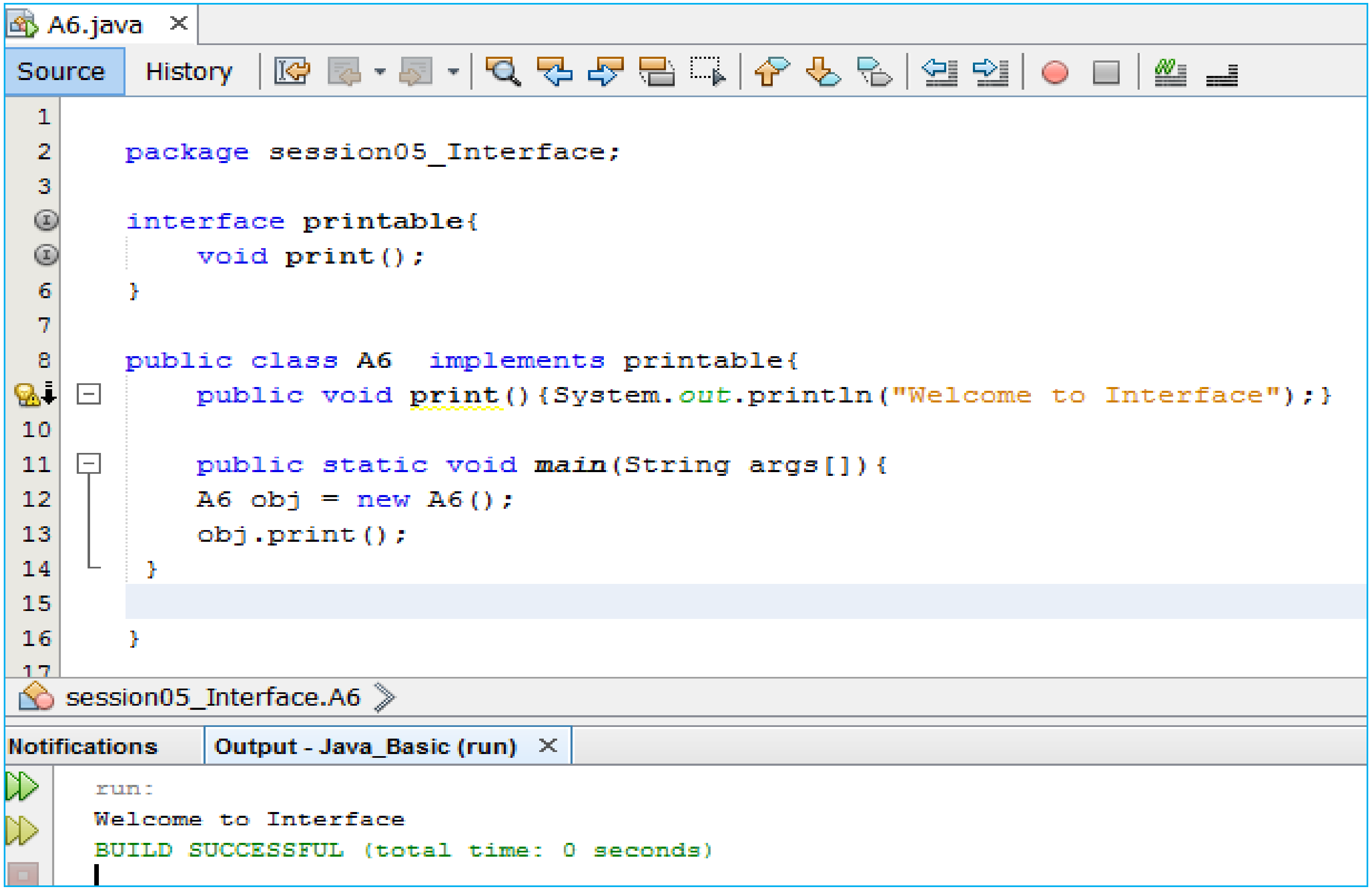
Interfaces

- An *interface* is a reference type, similar to a class, that can contain *only* constants, initialized fields, static methods, prototypes (abstract methods, default methods), and nested types.
- It will be the **core** of some classes
- Method bodies exist only for default methods and static methods.
- Interfaces cannot be instantiated because they have no-body methods.
- Interfaces can only be *implemented* by classes or *extended* by other interfaces.

Interfaces

Một **Interface trong Java** là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java. Interface trong Java cũng biểu diễn mối quan hệ IS-A. Nó không thể được khởi tạo giống như lớp trừu tượng.

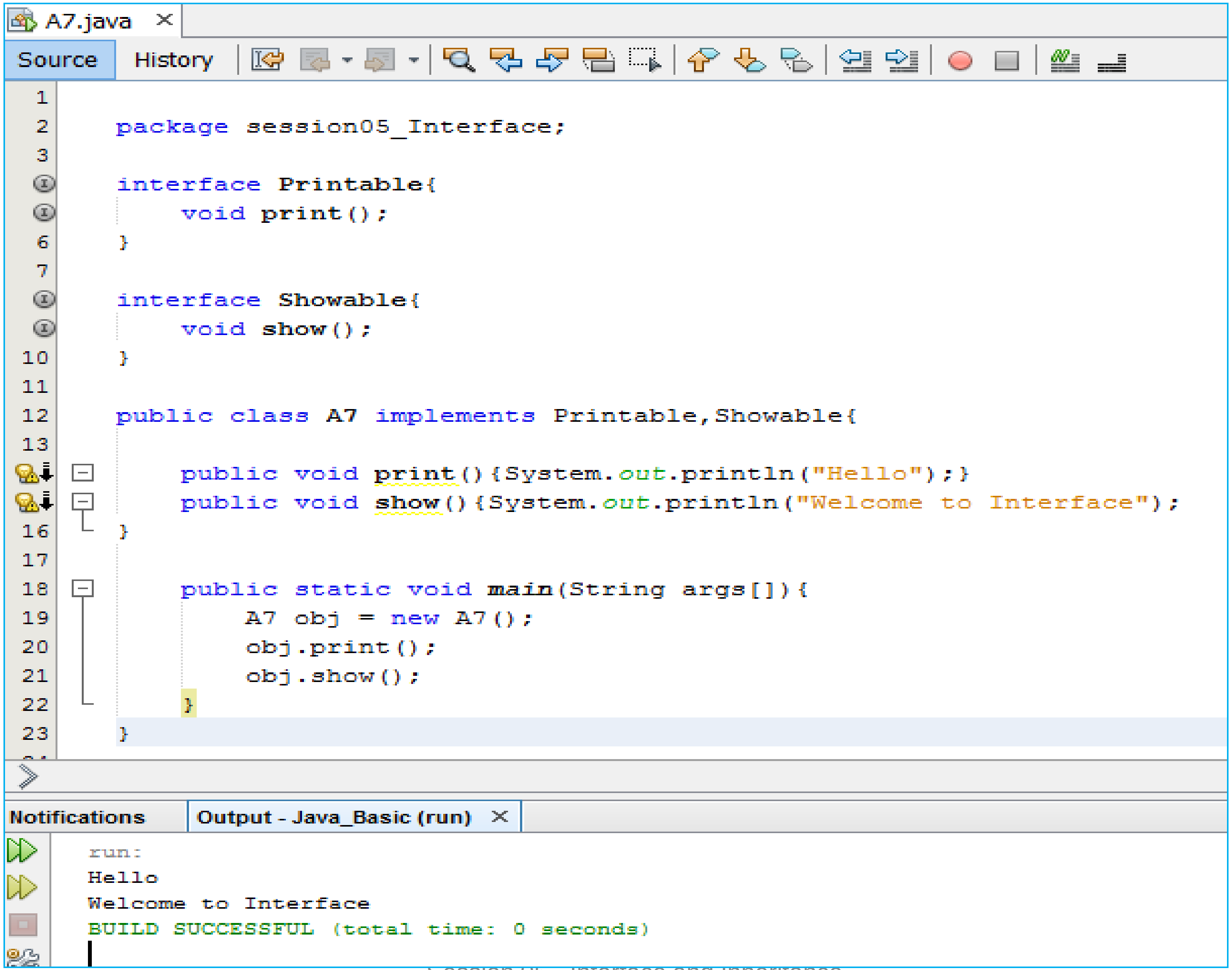
Interfaces



```
package session05_Interface;

interface printable{
    void print();
}

public class A6  implements printable{
    public void print(){System.out.println("Welcome to Interface");}

    public static void main(String args[]){
    A6 obj = new A6();
    obj.print();
 }

}
```

```
run:
Welcome to Interface
BUILD SUCCESSFUL (total time: 0 seconds)
```

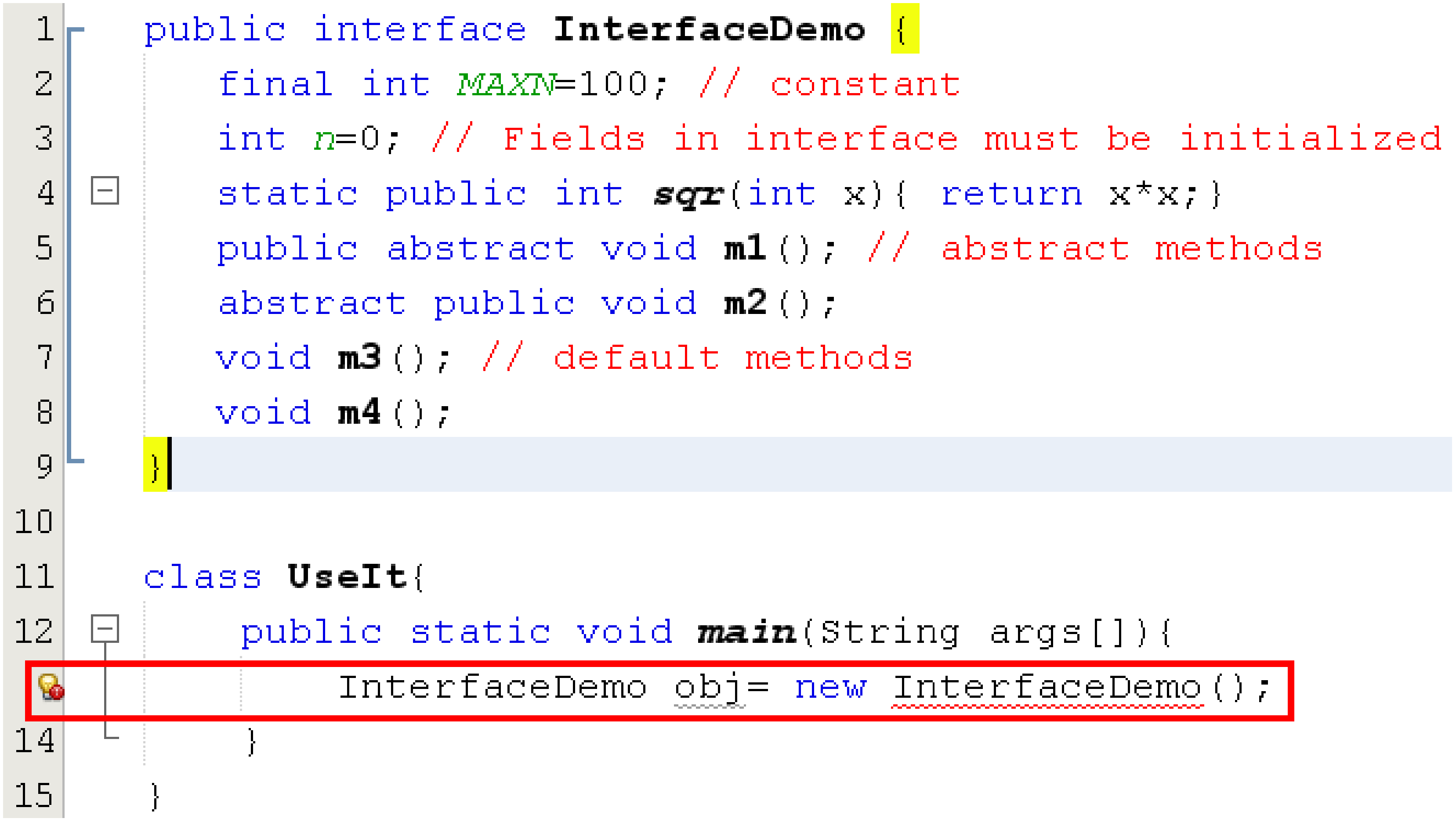
Interfaces



```java

package session05_Interface;

interface Printable{
    void print();
}

interface Showable{
    void show();
}

public class A7 implements Printable,Showable{

    public void print(){System.out.println("Hello");}
    public void show(){System.out.println("Welcome to Interface");
}

    public static void main(String args[]){
        A7 obj = new A7();
        obj.print();
        obj.show();
    }
}
```

```
run:
Hello
Welcome to Interface
BUILD SUCCESSFUL (total time: 0 seconds)
```

Interfaces…



```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}

class UseIt{
    public static void main(String args[]){
        InterfaceDemo obj= new InterfaceDemo();
    }
}
```

Interfaces…

```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}
class A implements InterfaceDemo{
    // overriding methods
    public void m1() { System.out.println("M1");}
    public void m2() { System.out.println("M2");}
    void m3() { System.out.println("M3");}
    void m4() { System.out.println("M4");}
}
```

m3(), m4() in A cannot implement m3(), m4() in InterfaceDemo, attempting to assign weaker access privileges, were public

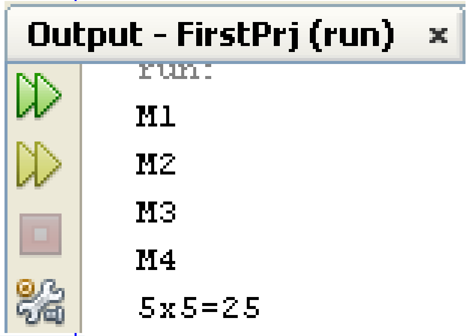
Default methods of an interface must be overridden as public methods in concrete classes.

Interfaces …

```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}
```

```
class A implements InterfaceDemo{
    // overriding methods
    public void m1() { System.out.println("M1");}
    public void m2() { System.out.println("M2");}
    public void m3() { System.out.println("M3");}
    public void m4() { System.out.println("M4");}
}
```

```
class UseIt{
    public static void main(String args[]){
        InterfaceDemo obj= new A();
        obj.m1();
        obj.m2();
        obj.m3();
        obj.m4();
        int s= InterfaceDemo.sqr(5);
        System.out.println("5x5=" + s);
    }
}
```



```
run:
M1
M2
M3
M4
5x5=25
```

Anonymous Classes

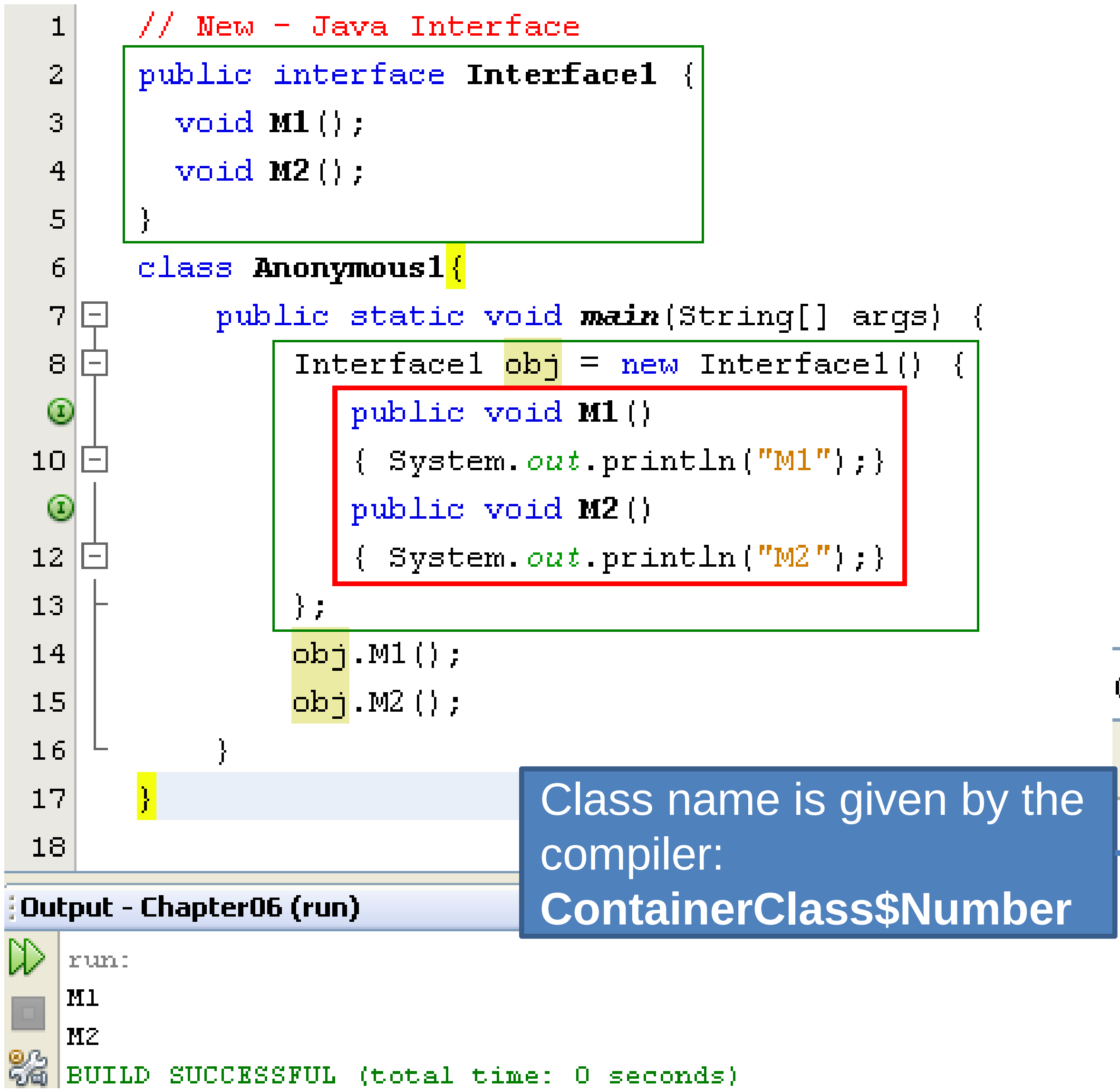
Anonymous classes are classes which are not named but they are identified automatically by Java compiler.

Where are they? They are identified at initializations of interface/abstract class object but abstract methods are implemented as attachments**.**

Why are they used?

- Enable you to make your code more concise.
- Enable you to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name.
- Use them if you need to use a local class only once.

Anonymous Class…



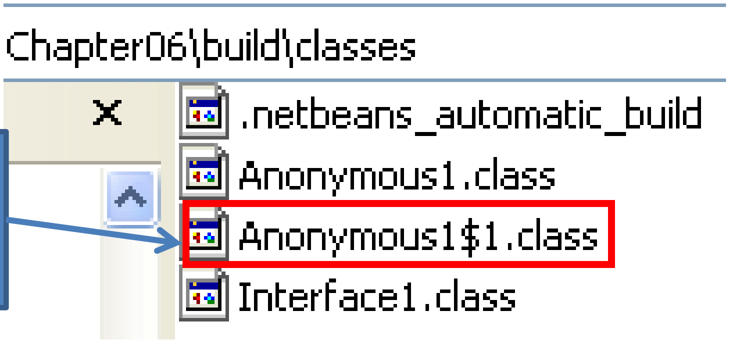
```java
// New - Java Interface
public interface Interface1 {
  void M1();
  void M2();
}
class Anonymous1{
    public static void main(String[] args) {
        Interface1 obj = new Interface1() {
            public void M1()
            { System.out.println("M1");}
            public void M2()
            { System.out.println("M2");}
        };
        obj.M1();
        obj.M2();
    }
}
```

Output - Chapter06 (run)

```
run:
M1
M2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Anonymous class.

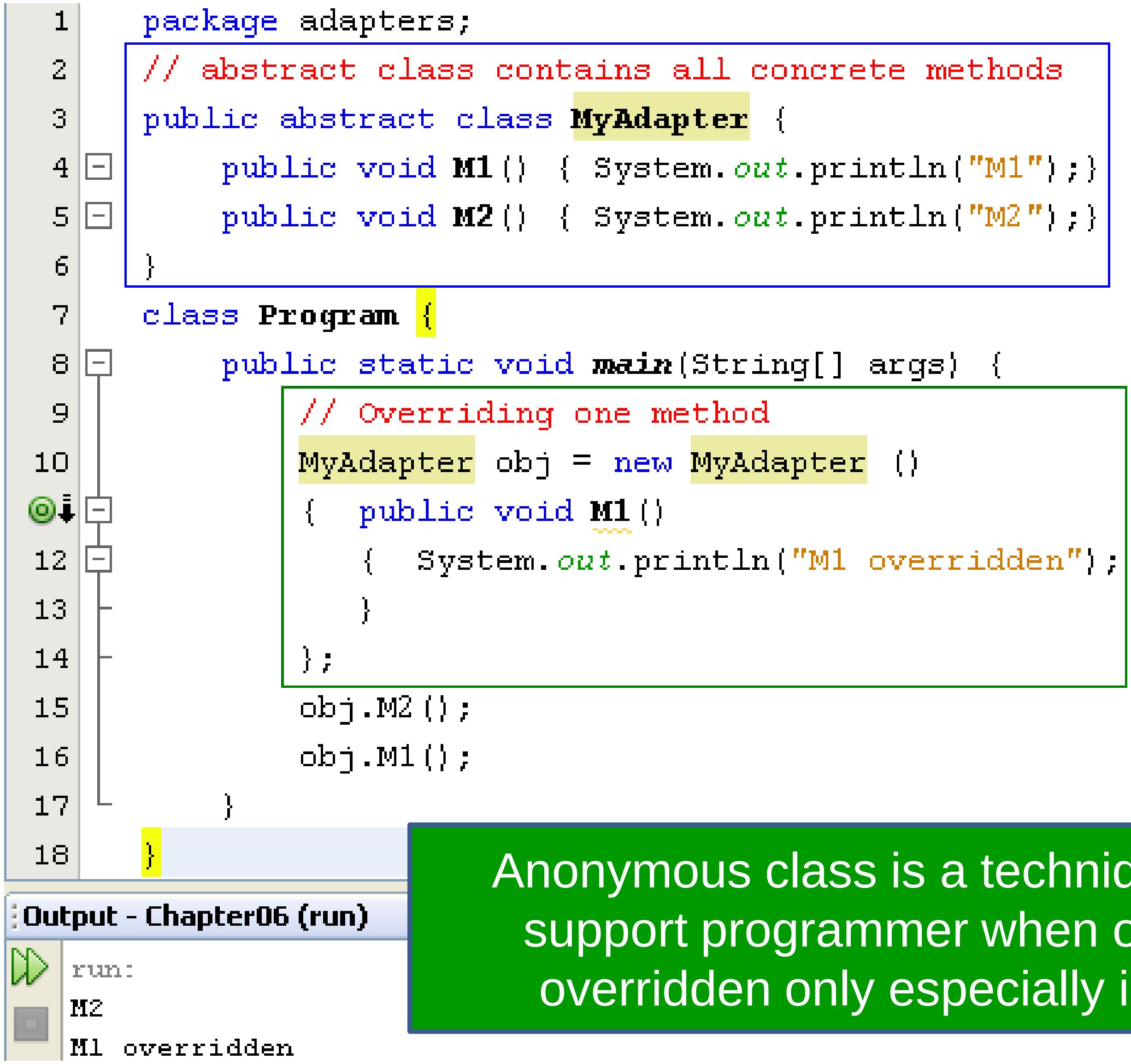
Class name is given by the compiler: **ContainerClass$Number**



Chapter06\build\classes

```
.netbeans_automatic_build
Anonymous1.class
Anonymous1$1.class
Interface1.class
```

Anonymous Class…



```
package adapters;
// abstract class contains all concrete methods
public abstract class MyAdapter {
    public void M1() { System.out.println("M1");}
    public void M2() { System.out.println("M2");}
}
class Program {
    public static void main(String[] args) {
        // Overriding one method
        MyAdapter obj = new MyAdapter ()
        {   public void M1()
            {   System.out.println("M1 overridden");
            }
        };
        obj.M2();
        obj.M1();
    }
}
```

Output - Chapter06 (run)

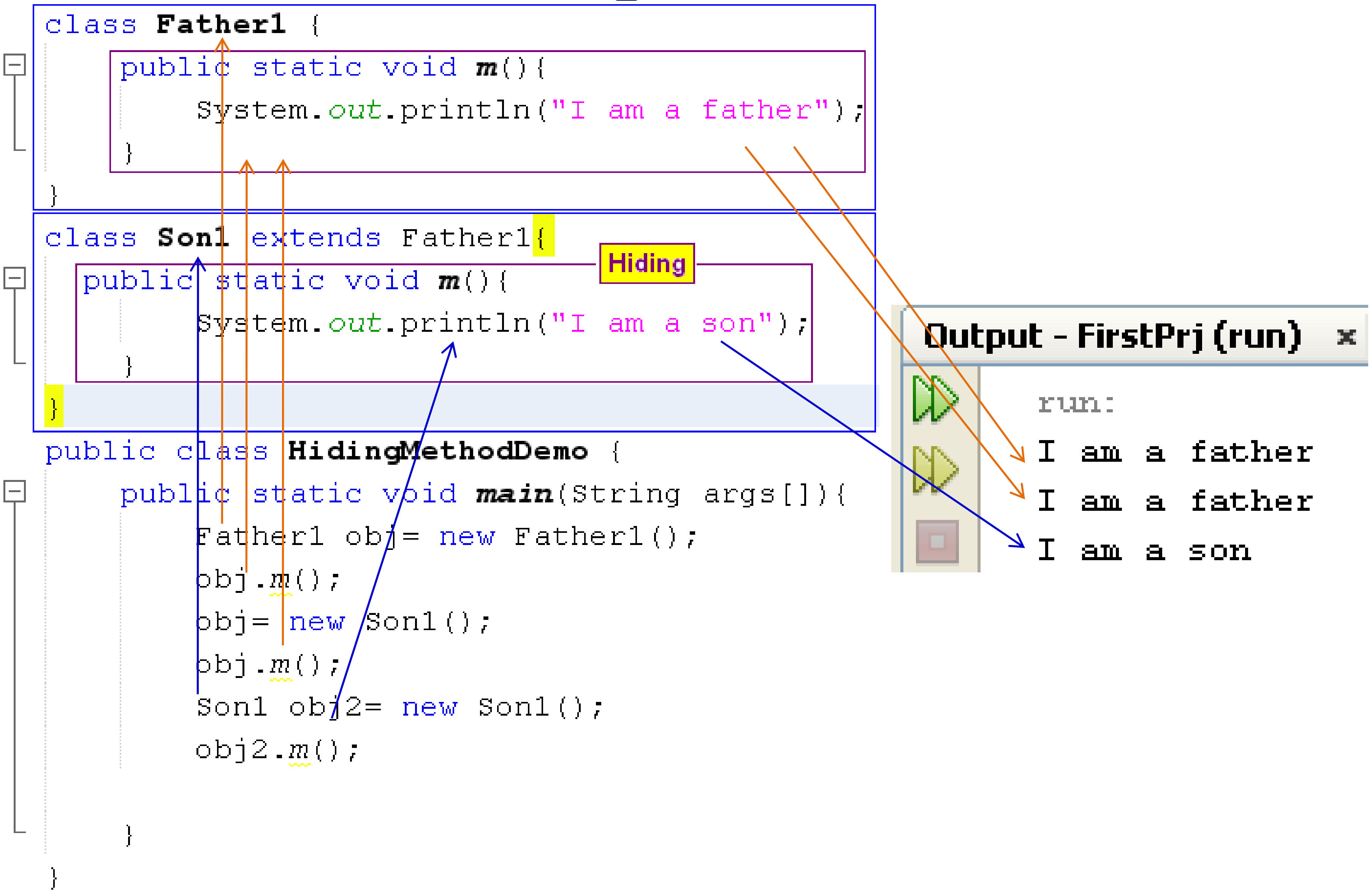
```
run:
M2
M1 overridden
```

Concrete methods but they can not be used because the class is declared as abstract one.

The abstract class can be used only when at least one of it’s methods is overridden

Anonymous class is a technique is commonly used to support programmer when only some methods are overridden only especially in event programming.

Hiding Methods

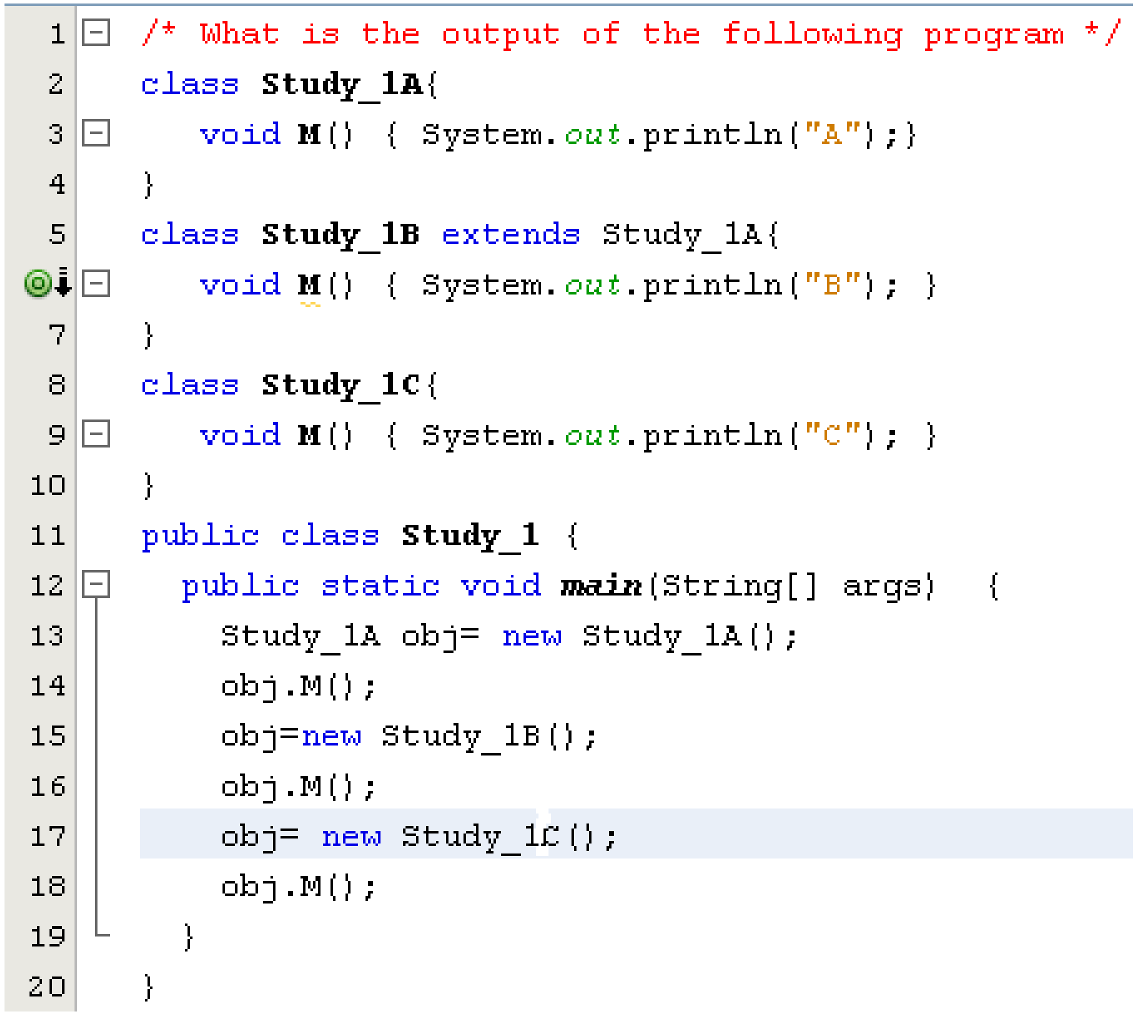


```
class Father1 {
    public static void m(){
        System.out.println("I am a father");
    }
}
class Son1 extends Father1{
    public static void m(){
        System.out.println("I am a son");
    }
}
public class HidingMethodDemo {
    public static void main(String args[]){
        Father1 obj= new Father1();
        obj.m();
        obj= new Son1();
        obj.m();
        Son1 obj2= new Son1();
        obj2.m();

    }
}
```

Output - FirstPrj (run)

```
run:
I am a father
I am a father
I am a son
```

Test



```java
/* What is the output of the following program */
class Study_1A{
    void M() { System.out.println("A");}
}
class Study_1B extends Study_1A{
    void M() { System.out.println("B"); }
}
class Study_1C{
    void M() { System.out.println("C"); }
}
public class Study_1 {
  public static void main(String[] args)   {
    Study_1A obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1C();
    obj.M();
  }
}
```

a) ABC

b) AAC

c) ABA

d) Compile-time error

Study_1A and Study_1C are inconvertible

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.println("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.println("B"); }
}
class Study_1C{
   void M() { System.out.println("C"); }
}
public class Study_1 {
  public static void main(String[] args)   {
    Object obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1C();
    obj.M();
  }
}
```

a) ABC

b) AAC

c) ABA

d) Compile-time error

The java.lang.Object class does not have the M() method

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.print("B"); }
}
class Study_1C{
   void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Study_1A obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    Object obj2= new Study_1C();
    ((Study_1A)obj2).M();
  }
}
```

a) ABC

b) AAA

c) ABA

d) None of the others

AB and a ClassCastException

Test

```
/* What is the following program */
class Study_1A{
   void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.print("B"); }
}
class Study_1C extends Study_1B {
   void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)   {
    Study_1A obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1C();
    obj.M();
  }
}
```

a) AAA

b) ACB

c) None of the others

d) ABC

Test

```
/* What is the output of the following program */
class Study_1A{
    void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
    void M() { System.out.print("B"); }
}
class Study_1C extends Study_1B {
    void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Study_1C obj= new Study_1C();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1A();
    obj.M();
  }
}
```

a) ABC

b) AAA

c) ABA

d) None of the others

Compile-time error
(Type conformity violation)

Test

```
public class Study_2 {
   static int N =10;
   int x = 120;
   static{
       N = 50;
       System.out.print("A");
   }
   public void M(){
       System.out.print(x);
   }
   public static void main(String [] args){
       Study_2 obj = new Study_2();
       obj.M();
   }
}
```

a) 120

b) 120A

c) None of the others

d) A120

Test

```
public class Study_2 {
  static int N =10;
  int x = 120;
  static{
      N = 7;
      System.out.print("A" + N );
      x = 500;
  }
  public void M(){
      System.out.print( x );
  }
  public static void main(String [] args){
      Study_2 obj = new Study_2();
      obj.M();
  }
}
```

a) A7500

b) 500A7

c) 500

d) None of the others

Compile-time error
(static code can not access instance variables)

Test

```
public class Study_2 {
   static int N = 2;
   int x = 10;
   static{
       N = 5;
       int y = 7;
       System.out.print("A" + (N + y) );
   }
   public void M(){
       System.out.print( x + y );
   }
   public static void main(String [] args){
       Study_2 obj = new Study_2();
       obj.M();
   }
}
```

a) A1210

b) 10A12

c) 17

d) None of the others

Compile-time error
(The y variable is out of scope)

Summary

- Polymorphism is a concept of object-oriented programming
- Polymorphism is the ability of an object to take on many forms
- Overloading and overriding are a technology to implement polymorphism feature.
- In OOP occurs when a parent class/ interface reference is used to refer to a child class object